Relationale Datenbanken - Typsystem, Modellierung und Objektorientierung

Prof. Dr. Klaus Meyer-Wegener

Fachvortrag auf der
3. Informatiklehrerkonferenz in Sachsen
(TU Dresden, Fakultät Informatik, 5. Juni 1999)

Datenbanken

was leisten sie? wann braucht man sie?

- Speicherung großer Datenmengen:
  - auch, aber nicht entscheidend
- **Datenunabhängigkeit** (der Anwendungen):
  - Benutzung der Daten, ohne Details der systemtechnischen Speicherung (einzelne Bits, Adressen) zu kennen (statt dessen abstraktes "Datenmodell", z.B. Tabellen)
  - einfache Handhabung der Daten, mächtige Zugriffs- und Auswertungsoperationen

Datenbanken (2)

- **Offenheit** der Daten für neue Anwendungen (Anwendungsneutralität der Speicherung):
  - symmetrische Organisationsformen (keine Bevorzugung einer Verarbeitungs- und Auswertungsrichtung)
  - explizite Darstellung der Annahmen (nicht in den Anwendungsprogrammen verstecken)
  - Redundanzfreiheit: keine wiederholte Speicherung in unterschiedlicher Form für verschiedene Anwendungen
  - Konsistenzüberwachung
  - Folge: *gemeinsame Nutzung* der Daten, auch gleichzeitig! (s. unten)

Datenbanken (3)

- **Ausfallsicherheit**
  - automatische Reparatur der Datenbestände nach Programm-, System- und Gerätefehlern
  - Rückgängigmachen unvollständiger Änderungen, so dass sie insgesamt wiederholt werden können
  - Wiederherstellen der Ergebnisse vollständiger Änderungen, so dass sie nicht wiederholt werden müssen

Datenbanken (4)

- **Mehrbenutzerbetrieb**
  - gleichzeitiger (zeitlich eng verzahnter) Zugriff verschiedener Anwendungen und Benutzer
  - Synchronisation,
    d.h. Steuerung der Reihenfolge von Zugriffen
    zur Vermeidung von Fehlern, die durch wechselseitige Beeinflussung auftreten können
  - Kooperation
    durch Verwendung gemeinsamer Daten
    mit hohem Aktualitätsgrad ("sofort")

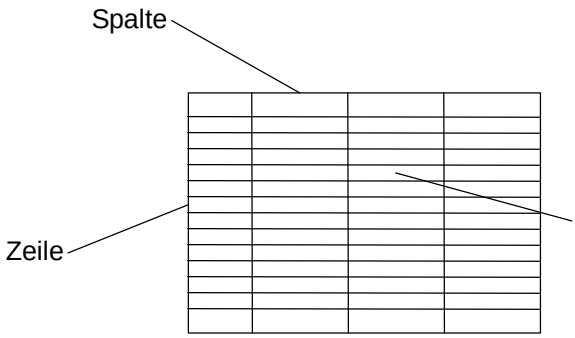
Relationale Datenbanken

- **Relation**
  - durchaus im mathematischen Sinne:
    Teilmenge eines Kartesischen Produkts

$$R \subseteq M_1 \times M_2 \times .... \times M_n$$

  - auch die Bezeichnung **Tupel** für die Elemente stammt aus der Mathematik (manchmal n-Tupel)
  - Mengen M_i prinzipiell beliebig,
    wichtig hier zunächst nur die Wertebereiche der gängigen Datentypen auf Rechnern

Relation bzw. Tabelle



atomar!
klein!
(Zeichenkette, Zahl, ...)

Darstellungsmittel

- **Relation / Tabelle**
- **Wertebereich / Domäne (domain), auch: Typ**
- **Attribut / Spalte (column)**
- **Tupel / Zeile (row)**
- **Primärschlüssel**
- **Fremdschlüssel**

mehr nicht!

Bewertung

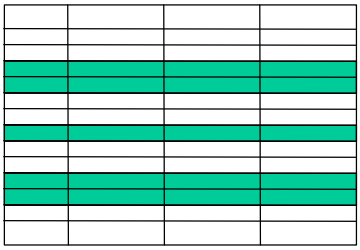
- nur Relationen sind speicherbar
- was immer man in der Datenbank ablegen will, muss die Form einer Relation haben
- es gibt nur den Typ Relation, wenn man Objekte erzeugen will - "in Atome zerlegen"

Anfragen, Auswertungen

- Relationenalgebra
- Relationenkalkül
- SQL

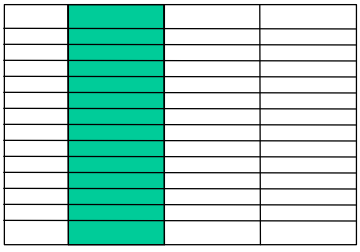
Selektion

select * from R where P(A)



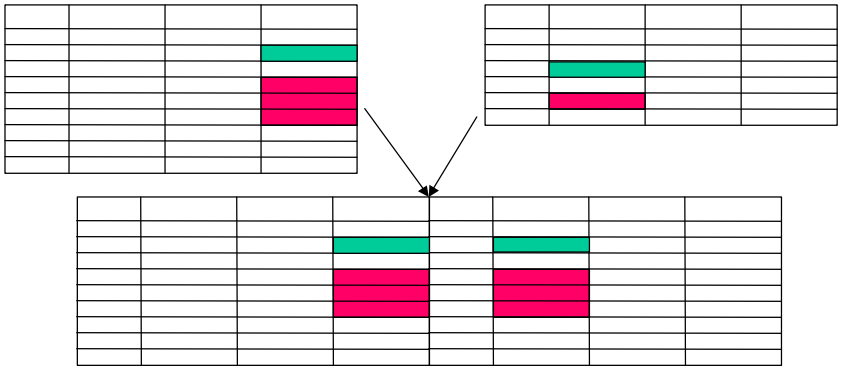
Projektion

select A from R



Verbund

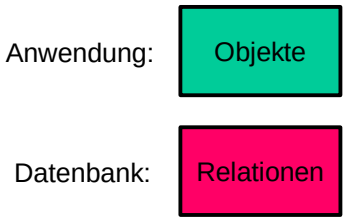
select * from R1, R2 where R1.A = R2.A



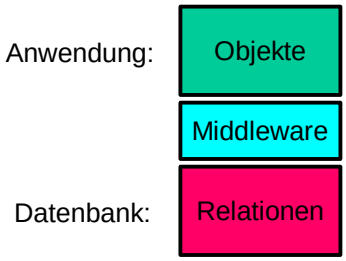
Neue Anforderungen

- komplexe Objekte in den Anwendungen
  - (z.B. 3D-Modelle, Landkarten, Bilder, ...)
- objektorientierte Anwendungsentwicklung
  - "Impedance Mismatch" beim Zugriff auf relationale Datenbanken
  - Brüche im Typsystem
  - Kopieren der Daten ("Auto in der Garage zerlegen")
- objektorientierte Datenbanken
  - Revolution statt Evolution, keine Migration

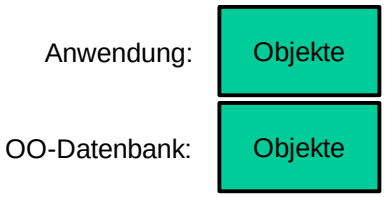
Lösungsmöglichkeit 1: "do it yourself"



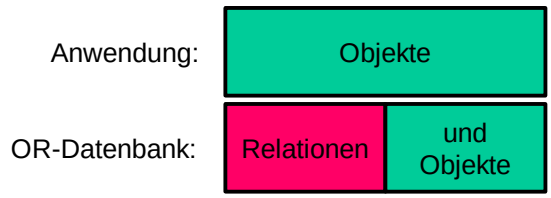
Lösungsmöglichkeit 2: Middleware für Objekte in DB



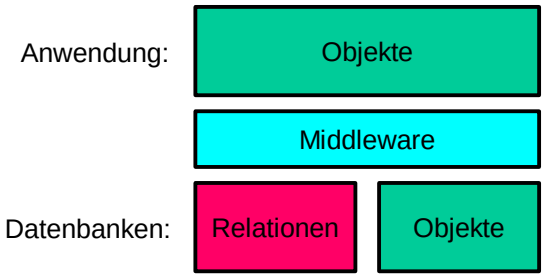
Lösungsmöglichkeit 3: objektorientierte DB



Lösungsmöglichkeit 4: objektrelationale DB



Lösungsmöglichkeit 5: föderierte (heterogene) DB



Objektrelationale DBS

- eingeführt mit UniSQL (W. Kim)
- popularisiert durch M. Stonebraker mit Illustra, dann an Informix verkauft
- heute klare Entwicklungsrichtung fast aller RDBVS (Oracle, DB2, Informix, Ingres)
- Normung: SQL-3
- Erweiterung des relationalen Datenmodells um OO-Konzepte bzw. Kombination beider Ansätze
  - Beibehaltung der bewährten Konzepte
  - Datenbank weiterhin Menge von Relationen

Erweiterbares Typsystem

- benutzerdefinierte Typen (user-defined types, UDT's) sowohl für Zeilen als auch für Spalten
  - abstrakte Datentypen (ADT's) mit typspezifischem Verhalten
  - Zeilentypen für hierarchische (geschachtelte) Strukturen
  - Verweis-Typen (reference)
- Typ-Konstruktoren
  - Definition von Sammlungen (collections) von Datentypen
  - Speicherung mehrerer Werte in einer Spalte einer Tabelle
  - Konstruktion geschachtelter Tabellen
- Vererbung

Beispiele: Spaltentyp

```
create function Name_Ein (text) returns Namenstyp
   as external name 'name.so' language C;
create function Name_Aus (Namenstyp) returns text
   as external name 'name.so' language C;
create type Namenstyp (
   internallength = 24,
   input = Name_Ein,
   output = Name_Aus );
```

Beispiele: Zeilentypen und Sammlungen

```
create type Beschreibungstyp (
   Kopf      varchar(50),
   Länge     integer,
   Rümpfe    setof(text) );
create table Einheit of new type Einheitentyp (
   Einheitenname      Namenstyp,
   Designer           varchar(20),
   Modellbauer        setof(varchar(20)),
   Tests              setof(Testtyp) );
```

Beispiele: Verweis-Typen

```
create table Anforderung on new type Anforderungstyp (
   Anforderungsname   Namenstyp,
   Beschreibung       Beschreibungstyp,
   Tests              setof(ref(Testtyp)) );
create table Test of new type Testtyp (
   Testname           Namenstyp,
   Testeinheit        ref(Einheitentyp),
   für_Anforderungen  setof(Anforderungstyp),
   Testdatum          date,
   Testmuster         array(array(Namenstyp)),
   sketch             Image );
```

Beispiele: Vererbung

```
create table Dymanischer_Test under Test (
    Beschleunigung          string,
    Amplitudensteigung      string,
    .... );
```

Beispiele: Anfragen

```
select from Automobil a where Gewicht_der_Sitze(a) < 300
select * from Anforderung
    where Beschreibung.Länge > 10000
select * from Test t
    where deref(t.Testeinheit).Designer = "Schmidt"
select Testname from Test t
    where "Schmidt" in deref(t.Testeinheit).Modellbauer
select * from Test t where Testdatum = "1997-03-12"
```

(schließt den Subtypen Dynamischer_Test ein)

SQL3 in der Entwicklung

- nur DB-Server-Erweiterungen, nichts für die Clients
- Defizite des Datenmodells:
  - Tabellen immer noch die einzigen Einheiten, die
    - persistent gespeichert werden können
    - in Anfragen angesprochen werden können
  - Zeilentypen (Row types) ohne Einkapselung
  - Verweistypen
    - nur anwendbar auf Zeilentypen
    - keine referentielle Integrität oder andere Semantik

SQL3 in der Entwicklung (2)

- ADTs als erster Schritt zur Objektorientierung, aber keine optimale Lösung
  - keine Objektidentifikatoren
  - ADT-Instanzen nur innerhalb einer Tabelle persistent speicherbar
  - keine Namen für einzelne ADT-Instanzen
  - keine Extension (Sammlung aller Exemplare) zu einem ADT

Zusammenfassung und Ausblick

- objektrelationale DBS breiten sich aus
  - aufwärtskompatibel
  - einfach mit dem nächsten Release
- Nutzung noch zurückhaltend
  - Entwurf sehr komplex und noch wenig untersucht
  - Performance
  - Optimierer: Forschungsthema
  - Warten auf SQL3
  - unverändert Impedance Mismatch
- dennoch: Weiterentwicklung und Dominanz